

Základní pojmy v algoritmickém myšlení: Sekvencování, výběr a opakování

Obsah

[Úvod](#_heading=h.gjdgxs)  **3**

[Co je to algoritmus?](#_heading=h.y9fd3ccnzb3t)  **3**

[Sekvence](#_heading=h.sx9mm8vv41lj)  **4**

[Výběr](#_heading=h.7a1rycfnc77v)  **6**

[Opakování](#_heading=h.c0s0rsaecwjw)  **7**

[Pseudokód](#_heading=h.8i64txlf9n76)  **9**

[Vyřešené problémy](#_heading=h.8f8zlhq795e6)  **11**

[Reference](#_heading=h.2et92p0)  **11**

[Přílohy](#_heading=h.tyjcwt)  **11**

# Úvod

Algoritmické myšlení je schopnost řešit problémy definováním jasného a definovaného sledu kroků, které, jakmile budou věrně dodrženy, vedou k řešení. Během školního života, a to zejména během let na základní a střední škole, se učíme řešit problémy procesem mnemotechnického učení a napodobováním řešení poskytovaných učitelem. Například, ačkoli mnoho studentů ví, jak vypočítat největšího společného jmenovatele dvou čísel, jen málokdo dokáže formulovat seznam operací potřebných k dosažení výsledku.

Tento proces definování kroků potřebných k dosažení řešení je nesmírně důležitý, protože kromě toho, že umožňuje automatizovat úkoly (např. vytvořením stroje schopného provádět uvedené kroky), vede jednotlivce k většímu povědomí o problému, jeho řešení a následně i schopnost přizpůsobit nalezené řešení podobným nebo analogickým problémům: aby bylo možné formulovat jasné a dobře definované kroky, je třeba nejprve porozumět problému, podmínkám, ze kterých vychází, a těm které si přejeme dorazit, a nástroje a operace, které nám umožňují modifikovat systém naprogramovaným a předvídatelným způsobem.

Přestože jsou pojmy „algoritmus“ a „algoritmické myšlení“ často bezprostředně spojeny s kontextem informatiky, jsou nepostradatelnými nástroji v každodenním životě. Stačí myslet na vaření a recepty používané k vaření jakéhokoli pokrmu, chirurgický zákrok nebo stavbu budovy. Definice algoritmů je jedním z procesů, které nejvíce využíváme k předávání znalostí a definovaných procesů.

V této práci bude podrobně popsán koncept algoritmu a blíže budou zkoumány některé z nejdůležitějších konceptů s ním souvisejících, jako je sekvenování, výběr a opakování.

# Co je to algoritmus?

Algoritmus, také často jednodušeji označovaný jako postup, je soubor jasných a dobře definovaných kroků, které lze použít k dokončení úkolu. Příkladem algoritmu, který nám může pomoci lépe porozumět konceptu, je recept na přípravu kávy pomocí moka konvičky. Ve své nejjednodušší verzi lze recept shrnout takto [[1]](#footnote-1):

* Rozložte moka konvičku na její tři hlavní součásti: spodní část, kovový filtr a horní část.
* Nalijte vodu do spodní části moka konvičky
* Umístěte kovový filtr do štěrbiny pro filtr na horní straně spodní části moka konvičky
* Naplňte kovový filtr kávovým práškem
* Našroubujte horní část moka konvičky na její spodní část
* Umístěte moka konvičku na sporák
* Zapněte sporák
* Počkejte, až káva vyteče, a naplňte horní část moka konvičky
* Vypněte sporák

Těmito pokyny se může efektivně řídit každý, i přiměřeně sofistikovaný robot, a vždy vedou ke stejnému výsledku: skvělé kávě.

Takže všechny seznamy operací jsou algoritmy? Algoritmus je ve skutečnosti na formální úrovni vždy charakterizován čtyřmi základními vlastnostmi: efektivitou, konečností výrazu, konečností výpočtu, determinismem.

* 1. **Účinnost**

Algoritmus musí být efektivně vykonatelný exekutorem; to znamená, že vykonavatel musí být schopen porozumět popisu algoritmu, a tudíž být schopen rozpoznat jazyk, ve kterém jsou kroky tvořící postup vyjádřeny. Dobře vytvořené věty v tomto zvoleném jazyce se formálně nazývají instrukce.

Jinými slovy, tato vlastnost znamená, že pokud definujete algoritmus pomocí jazyka, například angličtiny, každý exekutor schopný porozumět tomuto jazyku bude interpretovat instrukce stejným způsobem.

* 1. **Konečnost výrazu**

Postup popsaný algoritmem musí být vyjádřen v konečném počtu instrukcí. Nezáleží na tom, kolik instrukcí existuje (a lze snadno vymyslet procedury s velmi velkým počtem instrukcí), ani na délce procedury, pokud počet instrukcí není nekonečný.

* 1. **Konečnost výpočtu**

Podobně jako u konečnosti výrazu musí být algoritmus vždy charakterizován konečným počtem kroků provedení. Jinými slovy, algoritmus musí vždy specifikovat podmínku, pro kterou provádění končí. Zde opět není důležitý počet kroků, ale pojem konečnosti provedení.

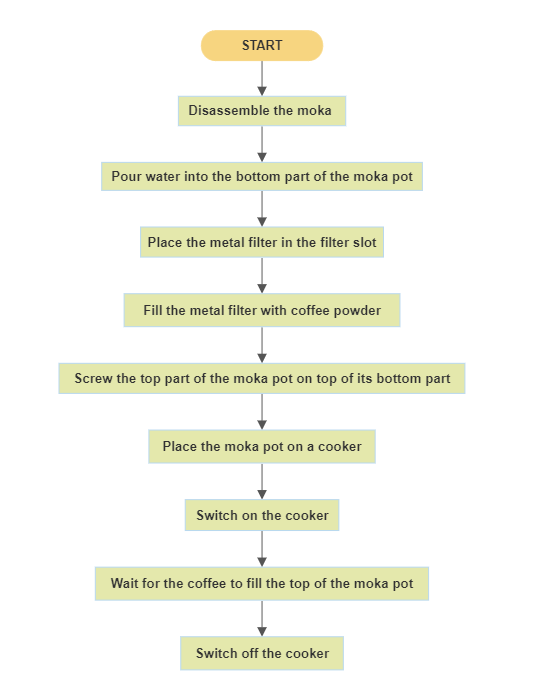
* 1. **Determinismus**

Algoritmus musí být deterministický, což znamená: použití stejného algoritmu na stejnou sadu vstupních dat vždy poskytne stejné výstupní výsledky.

Ve světle těchto čtyř vlastností si uvědomujeme, že je třeba učinit určité předpoklady, abychom zachovali náš recept na kávu jako algoritmus: vždy musí být přítomna čistá, sestavená moka konvička připravená k použití; vždy musí být k dispozici vařič; voda a kávový prášek musí být vždy k dispozici. Co kdybychom chtěli udělat náš algoritmus všestrannější, vzít v úvahu tyto počáteční podmínky a zobecnit tak, abychom získali postup, který je vždy proveditelný? V následujících kapitolách se blíže podíváme na některé nástroje, které nám umožní vylepšit náš algoritmus a zároveň lépe porozumět tomuto světu. Konkrétně uvidíme tři základní struktury: Sekvenční, Výběr a Opakování. Ty jsou ve skutečnosti nejdůležitější, zejména díky větě, kterou v 60. letech definovali dva badatelé: Corrado Böhm a Giuseppe Jacopini [1]. Věta říká, že s použitím pouze Senquencing, Selection a Repetition lze implementovat jakýkoli algoritmus!

# Sekvencování

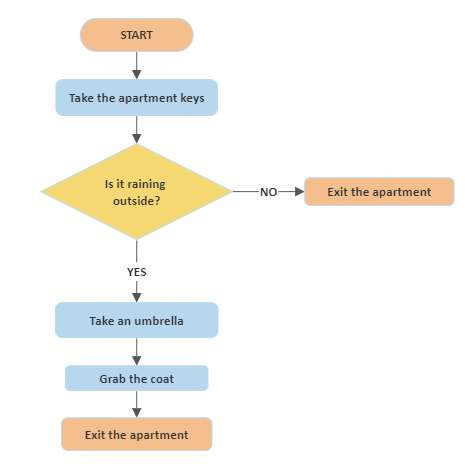
Sekvencování znamená definování pořadí pro instrukce, které jsou součástí algoritmu. Pokud například vezmeme v úvahu pokyny, které tvoří náš recept na přípravu kávy:



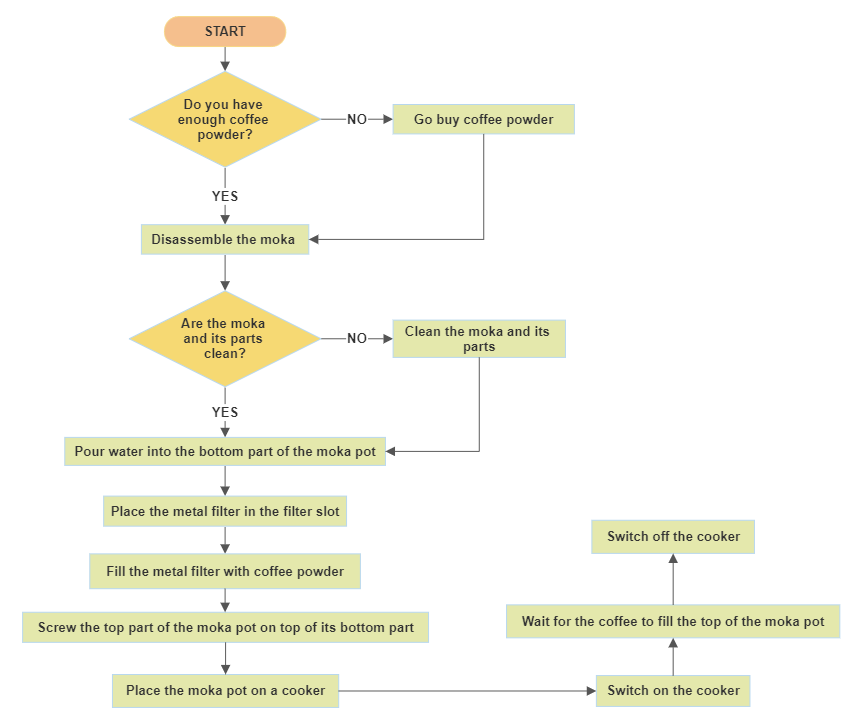
Aby bylo zajištěno, že bude vždy dosaženo stejného výsledku (zajištění determinismu), musí být každý krok proveden ve stanoveném pořadí. Podle tohoto příkladu je snadné vidět, že změna pořadí kroků by vedla k jinému (a nezamýšlenému!) výsledku: co by se stalo, kdybyste se pokusili naplnit spodní část moka konvičky vodou, než jste moka samotnou rozebrali?

# Výběr

Selekce je konstrukce, která nám umožňuje definovat různé cesty „provádění“ a zvolit jednu místo druhé v závislosti na ověření konkrétní podmínky (např. je podmínka pravdivá nebo nepravdivá?). Pokud bychom například definovali algoritmus pro shromažďování důležitých předmětů před odchodem z domu, mohli bychom definovat dvě cesty podle toho, zda venku zrovna prší nebo ne:



Výběr je velmi výkonný konstrukt, který nám umožňuje vytvořit algoritmus flexibilní a adaptabilní. Vezmeme-li jako příklad náš algoritmus pro přípravu kávy, můžeme začít testovat podmínky, abychom se ujistili, že je stále více použitelný v každodenní realitě: co se stane, když je konvice na kávu špinavá? Co se stane, když nám dojde kávový prášek?



# Opakování

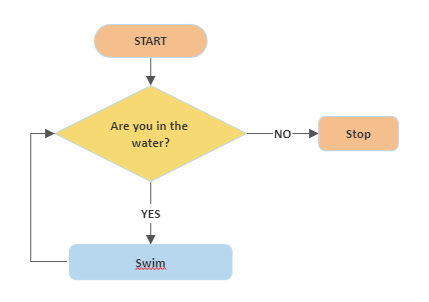
Opakování, často také nazývané iterace nebo smyčkování, je konstrukce, která umožňuje instrukci (nebo sadu instrukcí) provést více než jednou, aniž by bylo nutné je pokaždé opakovat v kontextu sekvenčního instrukčního bloku. Běžně existují tři typy opakování: *řízené počtem* (nazývané také „smyčka for“ kvůli klíčovým slovům používaným v různých programovacích jazycích k označení tohoto typu opakování), *while* a *repeat-until* .

* 1. **„Počítadlem řízené“ opakování**

Opakování řízené počtem je typ opakování, který umožňuje opakování bloku instrukcí předem definovaným počtem opakování. Například, pokud bychom chtěli explicitně uvést instrukce pro odpočítávání od 10 do 0, mohli bychom říci „odečíst 1 od aktuálního počítání 10krát za sebou“. Tento konstrukt je užitečný v kontextu operací, o kterých již předem víme, že se budou opakovat určitý počet opakování a jejichž opakování nezávisí na žádné další podmínce.

* 1. **„Zatímco“ opakování**

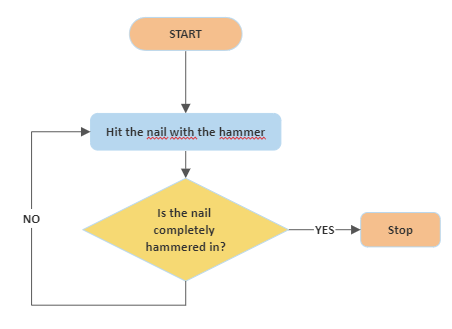
Na rozdíl od řízení počtu, zatímco opakování umožňuje opakování bloku instrukcí, dokud není splněna podmínka. Tento konstrukt umožňuje definovat skutečnost, že instrukce mají být provedeny několikrát, aniž bychom předem věděli, kolikrát se mají opakovat. Příklad tohoto typu popravy je následující: „plavání ve vodě“.



Je důležité poznamenat, že při použití opakování while nemáme žádnou jistotu, že instrukční blok, který se má opakovat, bude proveden alespoň jednou. Pokud je v uvedeném příkladu exekutor od začátku mimo vodu, pokyn „plavat“ se nikdy neprovede.

* 1. **Opakování „Opakovat do“.**

Opakovat-dokud opakování je ve skutečnosti velmi podobný konstrukt jako while: také se používá, když se blok instrukcí potenciálně musí opakovat několikrát a předem nevíme, kolikrát se bude muset opakovat. Jediný rozdíl je v tom, že kontrola stavu zastavení se provádí po první iteraci instrukce. Jinými slovy, zajistíme, aby byly instrukce provedeny alespoň jednou, než se rozhodneme, zda je zopakovat či nikoli.



# Pseudo kód

Vývojové diagramy jsou vynikajícím nástrojem pro formalizaci algoritmu ve srozumitelném a zapisovatelném jazyce. Jakkoli jsou však užitečné, mají důležité nevýhody, které je často činí nepohodlným při použití: množství požadovaného prostoru a okamžité mapování typických opakujících se konstrukcí (např. řízené počítání). Oblíbenou alternativou je ten, který poskytuje pseudokód: jazyk, který se podobá programovacím jazykům, ale abstrahuje z nich konkrétní klíčová slova, zůstává obecný a blíže přirozenému jazyku, který používáme každý den.

V této kapitole si stručně představíme zjednodušený pseudokód, který nám snadno umožní psát vlastní algoritmy pomocí námi představených konstrukcí. V tomto jazyce budeme používat následující pravidla:

* každý řádek představuje instrukci

|  |
| --- |
| toto je jeden pokyn  toto je jiný návod  toto je jiný návod |

* bloky instrukcí musí být reprezentovány použitím odsazení: různé bloky budou mít různý počet prázdných míst před řádkem

|  |
| --- |
| tato instrukce patří do **bloku A**  tato instrukce patří také do **bloku A**  tato instrukce patří do **bloku B**  tato instrukce také patří do **bloku B**  tato instrukce také patří do **bloku B**  tato instrukce patří do **bloku A** |

* Výběr je identifikován klíčovým slovem **IF** , následovaným podmínkou, která má být ověřena, a poté novým instrukčním blokem, který se má provést, pokud je podmínka pravdivá. Po tomto bloku lze použít další blok instrukce k definování instrukce, která se má provést, pokud je podmínka nepravdivá; tento blok musí být uvozen klíčovým slovem **ELSE** .

|  |
| --- |
| pokyn  jiný pokyn  **IF podmínka**  **tato instrukce bude provedena, pokud je podmínka pravdivá**  **tato instrukce bude také provedena, pokud je podmínka pravdivá**  **JINÝ**  **tato instrukce bude provedena, pokud je podmínka nepravdivá**  **tato instrukce bude také provedena, pokud je podmínka nepravdivá**  tato instrukce bude provedena bez ohledu na podmínku uvedenou po IF |

* Počet řízené opakování je označeno klíčovým slovem **FOR** **X TIMES DO** následovaný instrukčním blokem, který se má Xkrát opakovat.

|  |
| --- |
| pokyn  jiný pokyn  **X KRÁT UDĚLEJTE**  **tato instrukce se bude Xkrát opakovat**  **tato instrukce se bude také Xkrát opakovat**  tento pokyn se nebude opakovat |

* Zatímco opakování je identifikováno klíčovým slovem **WHILE <CONDITION> DO** následovaným instrukčním blokem, který se má opakovat, dokud <CONDITION> přestane platit. Pamatujte, že pokud podmínka není zpočátku pravdivá, instrukce v následujícím bloku nebudou provedeny.

|  |
| --- |
| pokyn  jiný pokyn  **ZA STAVU DO**  **tato instrukce se bude opakovat, dokud podmínka neplatí**  **tato instrukce se bude také opakovat, dokud podmínka neplatí**  tento pokyn se nebude opakovat |

* Opakovat-dokud je opakování identifikováno klíčovým slovem **REPEAT** následovaným instrukčním blokem. Po tomto instrukčním bloku musí následovat klíčové slovo **UNTIL <CONDITION>** , které specifikuje podmínku, která má být zkontrolována, aby bylo možné určit, zda se má instrukční blok opakovat nebo ne.

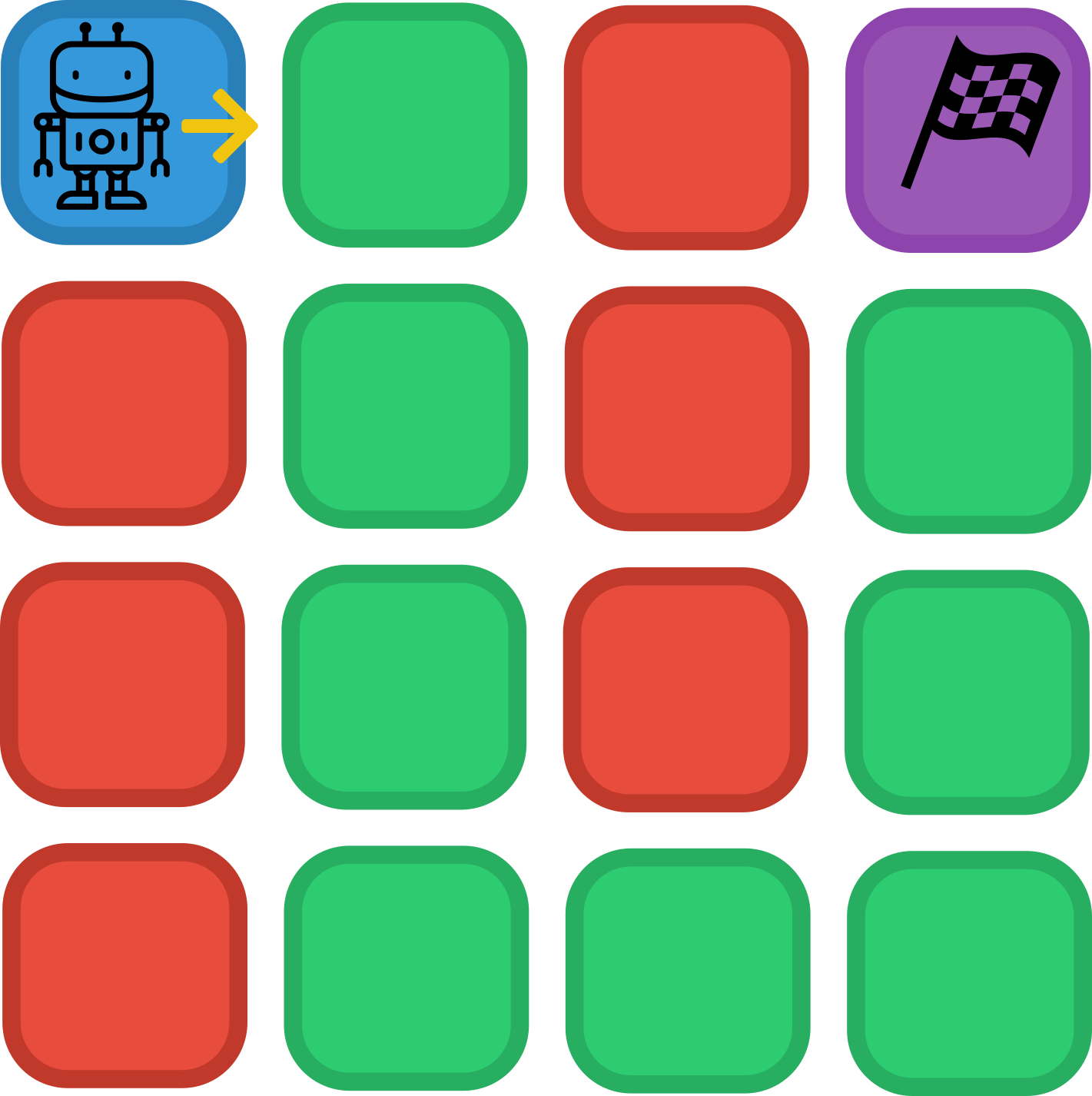
|  |
| --- |
| pokyn  jiný pokyn  **OPAKOVAT**  **tato instrukce se bude opakovat, dokud podmínka neplatí**  **tato instrukce se bude také opakovat, dokud podmínka neplatí**  **DO STAVU**  tento pokyn se nebude opakovat |

# Vyřešené problémy

1) Bob je malý robot. Protože je velmi jednoduchý, provádí pouze 3 operace:

* odbočte doleva o 90°
* odbočte vpravo o 90°
* jít o jednu dlaždici vpřed

Můžete podle následujícího obrázku pomoci Bobovi dostat se z modrého na fialový a pohybovat se pouze po zelených? Napište dva algoritmy pro tento úkol, jeden bez a druhý s použitím opakovacích konstrukcí. Všimněte si prosím, že žlutá šipka ukazuje směr, kterým se Bob začíná dívat.



|  |  |
| --- | --- |
| **Řešení bez opakování** | **Řešení s využitím opakovacích konstrukcí** |
| jít o jednu dlaždici vpřed  odbočte vpravo o 90°  jít o jednu dlaždici vpřed  jít o jednu dlaždici vpřed  jít o jednu dlaždici vpřed  odbočte doleva o 90°  jít o jednu dlaždici vpřed  jít o jednu dlaždici vpřed  odbočte doleva o 90°  jít o jednu dlaždici vpřed  jít o jednu dlaždici vpřed  jít o jednu dlaždici vpřed | jít o jednu dlaždici vpřed  odbočte vpravo o 90°  NA 3KRÁT UDĚLEJTE  jít o jednu dlaždici vpřed  odbočte doleva o 90°  PRO 2KRÁT UDĚLEJTE  jít o jednu dlaždici vpřed  odbočte doleva o 90°  NA 3KRÁT UDĚLEJTE  jít o jednu dlaždici vpřed |

# Reference

[1] Böhm, C., & Jacopini, G. (1966). Vývojové diagramy, Turingovy stroje a jazyky pouze se dvěma formačními pravidly. *Commun. ACM, 9* , 366-371.

[2] druhý odkaz

1. Ikona moka konvice vytvořená Dooderem a stažená z Flaticonu – „https://www.flaticon.com/free-icons/moka-pot“ [↑](#footnote-ref-1)